

# CDAT Utilities 3.3 – CHAPTER 3

## CHAPTER 3

### User Contributed Packages

The packages described below are contributions submitted by users. They are provided "as-is" and may not be maintained in the future – unless they are extensively used and the user community considers them critical.

#### *Reading ASCII text files (package asciidata)*

Package asciidata reads data from ASCII text files.

Reads text files written by such programs as spreadsheets, in which data has been written as comma, tab, or space-separated numbers with a header line that names the fields. Using the functions in asciidata, you can convert these columns into Numerical arrays, with control over the type/precision of these arrays.

#### Example

```
>>> import asciidata  
  
>>> time, pressure = asciidata.comma_separated('myfile.txt')
```

#### For documentation type:

```
% pydoc -w asciidata
```

Scientific Python also contains a subpackage IO that contains other useful facilities of this type. In particular there is a useful package for reading Fortran-like formatted output.

---

#### *Reading binary data (package binaryio)*

Read and write Fortran unformatted i/o files.

These are the files that you read and write in Fortran with statements like read(7) or write(7). Such files have an unspecified format and are platform and compiler dependent. They are NOT portable. Contrary to popular opinion, they are NOT standard. The standard only specifies their existence and behavior, not the details of their implementation, and since there is no one obvious implementation, Fortran compilers do vary. We suggest writing netcdf files instead, using the facilities in cdms.

#### For documentation type:

```
% pydoc -w binaryio.
```

A similar package is in Scientific Python.

Example:

```
>>> import binaryio  
  
>>> iunit = binaryio.bincreate('filename')  
  
>>> binaryio.binwrite(iunit, some_array)  
  
#(up to 4 dimensions)  
  
>>> binaryio.binclose(iunit)  
  
>>> iunit = binaryio.binopen('filename')  
  
>>> y = binaryio.binread(iunit, n, ...)  
  
# (1–4 dimensions)  
  
>>> binaryio.binclose(iunit)
```

Note that reads and writes must be paired exactly. Errors will cause a Fortran STOP that cannot be recovered from. You must know (or have written earlier in the file) the sizes of each array. All data is stored as 32-bit floats.

***Explicit Orthonormal Functions (package eof)***

Calculates Explicit Orthonormal Functions of either one variable or two variables jointly.

Having selected some data, the key call is to create an instance of `eof.Eof` giving one or two arguments. In this example, a portion of the variable `'u'` is analyzed. After the result is returned, it is an object with attributes containing such things as the principal components and the percent of variance explained. Optional arguments are available for controlling the subtraction of the mean from the data, the weighting by latitude, and the number of components to compute.

This routine is computationally efficient, solving the problem in either the normal space or the dual space in order to minimize computations. Nonetheless, it is possible that this routine will require substantial time and space if used on a large amount of data. This cost is determined by the smaller of the number of time points and the total number of space points.

For documentation type:

```
% pydoc -w eof.Eof
```

Example:

```
>>> import cdms, vcs  
  
>>> from eof import Eof
```

```

>>> f=cdms.open('/home/dubois/clt.nc')

>>> u = f('u', latitude=(-20,40), longitude=(60, 120))

>>> result = Eof(u)

>>> principal_components = result.principal_components

>>> print "Percent explained", result.percent_explained

>>> x=vcs.init()

>>> print len(principal_components)

>>> for y in principal_components:

>>> x.isofill(y)

>>> x.clear()

>>> u1 = v.subRegion(latitude=(amr[0], \
amr[1], 'cc'), longitude=(amr[2], \
amr[3],'cc'), order='xyt')

>>> result2 = Eof(u, number_of_components=4, \
mean_choice=12)

>>> print "Percent explained", result.percent_explained

```

### ***Computing L-moments (package lmoments)***

An interface to an L-moments library by J. R. M. Hosking.

This package is an interface to a Fortran library. The calling sequence from Python differs from the Fortran convention. In general, output and temporary arguments are not supplied in making the Python call, and output arguments are returned as values of the function.

#### **For documentation type:**

% pydoc -w lmoments

to see list of functions.

% pydoc -w lmoments.pelexp

or other function name, for the particular. See also documentation for Pyfort at [pyfortran.sourceforge.net](http://pyfortran.sourceforge.net) for further details on argument conventions. If built from source, a file flmoments.txt appears which gives the

Python calling sequences.

### ***Regridding using package regridpack***

Interface to regridpack

For documentation type:

```
% pydoc -w adamsregrid
```

This package contains a Python interface to the subroutine library regridpack.

Documentation online at cdat.sourceforge.net. See also documentation for Pyfort at pyfortran.sourceforge.net for further details on argument conventions.

### ***Using Sphrepak (package sphere)***

Interface to Sphrepak. This package contains a Python interface to the subroutine library Sphrepak.

For documentation type:

```
% pydoc -w sphere
```

to see list of functions.

Documentation online at cdat.sourceforge.net. See also documentation for Pyfort at pyfortran.sourceforge.net for further details on argument conventions.

### ***Computing Trends (package trends)***

Computes variance estimate taking auto-correlation into account.

Example:

```
import reg_arl from trends
```

```
rneff, result, res, cxx, rxx = reg_arl (lag, x, y)
```

integer lag Max lag for autocorrelations.

real x(n1) Independent variable

real y(n1) Dependent variable

```
real, intent(out):: rneff !Effective sample size  
real, intent(out):: result(31) !Array of linear regression results  
real, intent(out):: res(n1) !Residuals from linear regression  
real, intent(out):: cxx(1 + lag) !Autocovariance function  
real, intent(out):: rxx(1 + lag) !Autocorrelation function
```

### ***Reading data from an Oort file (package ort)***

Read data from an Oort file.

Module ort contains one Fortran function, read1f:

#### **Calling sequence:**

```
>> import ort  
>>> lon, lat, data, nr = ort.read1f(filename, maxsta,\  
nvarbs, nlevels)
```

#### **Input:**

```
character*(*) filename ! name of the file to be read  
! max number of stations (soundings) possible  
integer maxsta  
! number of variables and P-levels in each sounding  
integer nvarbs, nlevels
```

#### **Output:**

```
! longitudes / latitudes of the stations  
real, intent(out):: lon(maxsta), lat(maxsta)  
! sounding data  
real , intent(out):: data(nvarbs, nlevels, maxsta)
```

! actual number of stations with data

integer , intent(out):: nr

### ***A grads like interface (package grads)***

The grads module supplies an interface to cdms that will be familiar to users of GrADS.

See the CDAT website for documentation.

### ***Interface to the ngmath library. (package ngmath)***

The ngmath library is a collection of interpolators and approximators for one-dimensional, two-dimensional and three-dimensional data. The packages, which were obtained from NCAR, are:

- natgrid – a two-dimensional random data interpolation package based on Dave Watson's nngriddr. NOT built by default in CDAT due to compile problems on some platforms. Works on linux.
- dsgrid – a three-dimensional random data interpolator based on a simple inverse distance weighting algorithm.
- fitgrid – an interpolation package for one-dimensional and two-dimensional gridded data based on Alan Cline's Fitpack. Fitpack uses splines under tension to interpolate in one and two dimensions. NOT IN CDAT.
- csagrid – an approximation package for one-dimensional, two-dimensional and three-dimensional random data based on David Fuler's Splpack. csagrid uses cubic splines to calculate its approximation function.

## **A**

- [autocorrelation](#) 33
- [autocovariance](#) 37
- [averager](#) 1

## **B**

- [binaryio \(package\)](#) 52

## **C**

- [cdutil](#) 1, 6
- [centroid function](#) 11
- [correlation](#) 33
- [covariance](#) 35
- [criteriaarg](#) 10
- [custom seasons](#) 9

## **D**

- [data, Fortran binary](#) 52
- [data, Oort](#) 56
- [data, reading ASCII](#) 51

E

- [eof](#) 53

F

- [Fortran-like I/O](#) 52

G

- [Generating weights](#) 6
- [geometricmean](#) 44
- [grads \(GrADS-like interface\)](#) 57
- [grads \(module\)](#) 57
- [grower](#) 49

L

- [laggedcorrelation](#) 38
- [laggedcovariance](#) 39
- [linearregression](#) 45
- [lmoments](#) 54

M

- [meanabsdiff](#) 40
- [median](#) 45
- [minmax](#) 48

N

- [ngmath](#) 57

O

- [Oort data](#) 56
- [ort \(package\)](#) 56

P

- [percentile](#) 44
- [Pyfort](#) 55

R

- [reading files written by Fortran](#) 52
- [regridpack \(package\)](#) 55
- [rgb2str](#) 49

- rms 41

S

- Scientific Python 52
- sphere (package) 55
- spherepack module 55
- std 42
- str2rgb 50
- supporting objects 13

T

- Time averaging 8
- times tutorial.py 12
- trends (package) 56

V

- VariableConditioner 13
- VariableConditioner Object 19
- VariablesMatcher 12
- VariablesMatcher Object. 23
- variance 43
- variance w/auto-correlation 56

W

- WeightedGridMaker 13
- WeightedGridMaker Object 16
- WeightsMaker 13
- WeightsMaker Object 13
- writing Fortran unformatted files 52

X

- xmgrace (package) 48

[Go to Previous](#) | [Go to Main](#)